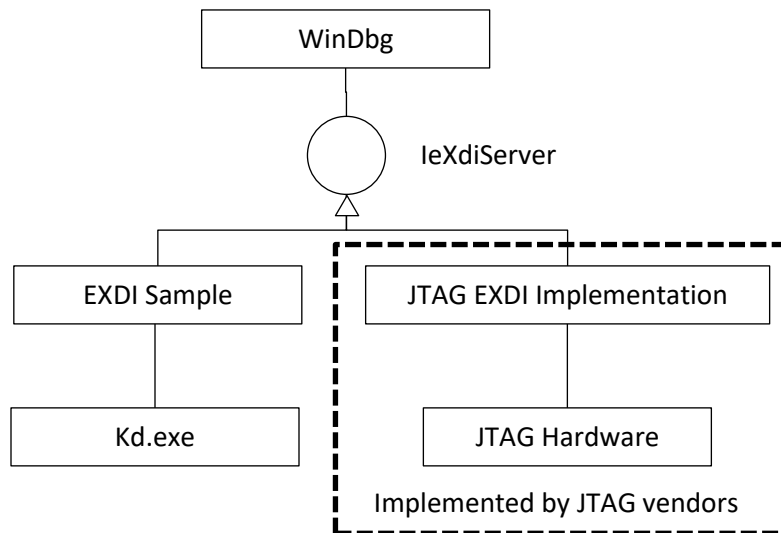


EXDI KD Sample – getting started

EXDI is an interface that allows extending WinDbg by adding support for hardware debuggers (e.g. JTAG-based). This sample is intended for JTAG debugger vendors that want to add support for their hardware to WinDbg and other Microsoft debuggers. The diagram below illustrates the role of EXDI:



The sample consists of 2 parts:

- A “static” sample. It supports viewing the state of a stopped target (e.g. view stack/variables/modules) and does not support resuming or setting breakpoints.
- A “live” sample. Additionally to all functionality of the static sample it supports stepping, setting breakpoints and resuming target execution.

Static EXDI sample

In order to make WinDbg support a third-party JTAG debugger in the “static mode” (analyze the state of a stopped target, no support for resuming or setting breakpoints) the JTAG vendor needs to provide an implementation of the EXDI interface supporting methods that:

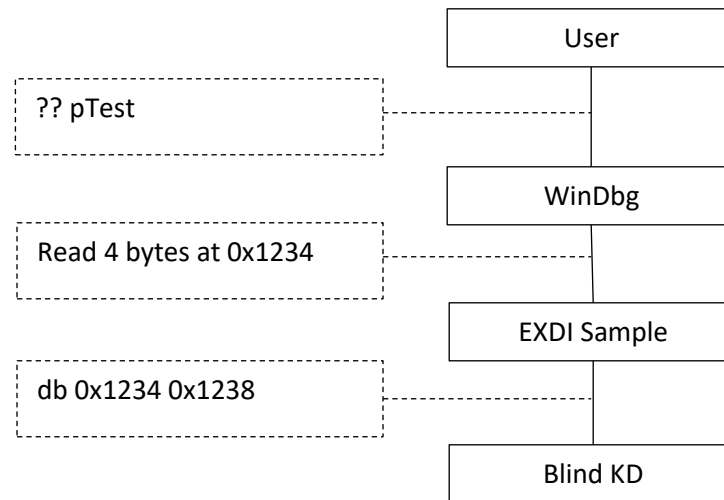
- Read virtual memory at a given address
- Read CPU registers

This is enough to provide debugging experience similar to analyzing crash dumps – WinDbg will handle symbols, unwind stacks and parse OS-specific structures.

This example does not depend on any real hardware. Instead we “emulate” a JTAG device by running command-line kernel debugger (kd.exe) and artificially restricting it to 2 basic commands:

- The ‘db’ command to read memory
- The ‘r’ command to read registers

We refer to the restricted kd.exe as “blind KD”. The example demonstrates how higher-level commands (e.g. evaluating a C++ expression) will be translated by Microsoft debugger engine into series of low-level commands, such as ‘read memory’ and handled by the EXDI implementation:



JTAG vendors should implement those basic operations using their JTAG programmers using this example as a reference.

This document assumes that the reader has basic experience debugging Windows drivers. If not, please refer to MSDN and WDK documentation for instructions on building and debugging a basic driver. It is recommended to deploy the driver on a virtual machine (e.g. Hyper-V).

Before you begin

Before starting to do anything with this example, please follow the preparation steps below:

1. Install Debugging Tools for Windows. It is recommended to use the 32-bit version of the tools.
2. Build the ExdiKdSample.sln solution and register ExdiKdSample.dll produced by the build by running ‘regsvr32 ExdiKdSample.dll’ as Administrator.
3. Create a driver project containing the following code:

```
#include <wdm.h>

extern "C" NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject,
PUNICODE_STRING RegistryPath)
{
    (void)DriverObject, (void)RegistryPath;

    const char *pTest = "Hello, World\n";
    DbgPrint("%s", pTest);
    DbgBreakPoint();
    return STATUS_NOT_IMPLEMENTED;
}
```

4. Build the driver, deploy it on a second machine (e.g. a Hyper-V virtual machine) and ensure that you can debug it with WinDbg. Note the command-line arguments used to launch WinDbg (e.g. `-k com:pipe,port=\\.\pipe\vmkerneltest1`).

5. Take a note of the **KdVersionBlock** address in the kernel you are about to debug. Connect to the kernel using normal WinDbg, break-in and run the following command:

```
kd> ? KdVersionBlock
Evaluate expression: -8788337193488 = fffff801`ce488df0
```

You will need the underlined decimal value later. Decimal is used for compatibility reasons.

Analyzing the system state with EXDI

We will now start a normal kernel debugging session with WinDbg, stop the kernel at a certain point, disconnect WinDbg and reconnect using EXDI. This document will explain how work is split between WinDbg (handling symbols) and the EXDI Server (fetching memory/registers). Please follow the steps below to get started:

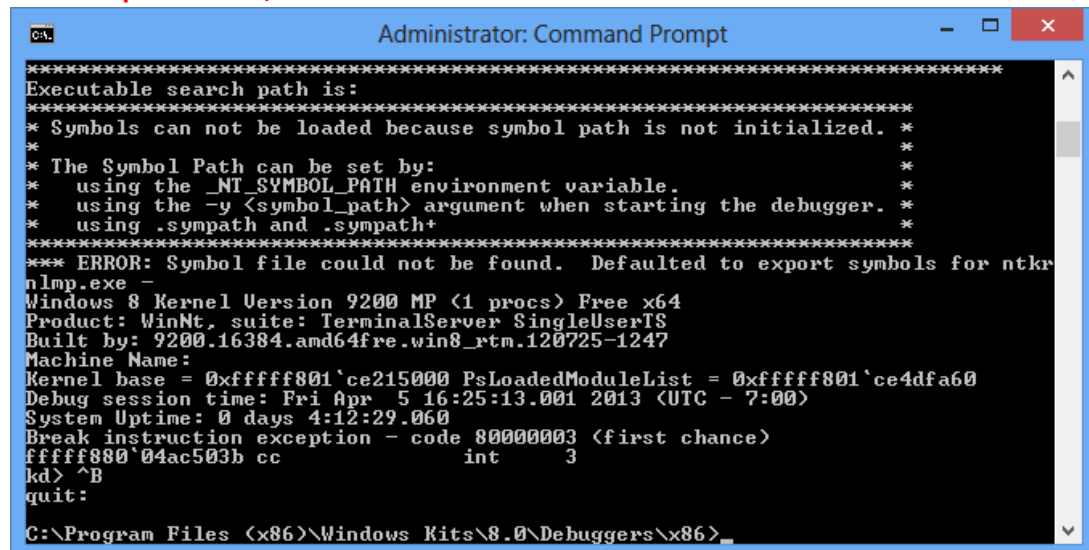
1. Start a normal kernel debugging session with WinDbg. Load the driver and wait until it stops on the **DbgBreakPoint()** call.
2. Ensure you have recorded the **KdVersionBlock** address.
3. Open a new Command Prompt window in Administrator mode and go to the Debugging Tools directory.
4. Try running kd.exe manually. E.g. when debugging an ARM tablet over USB having 'surface' as the debug target name, run the following command:

```
kd -k usb:targetname=surface
```

If you are debugging a virtual machine with COM1 redirected to a pipe called 'vmpipe', run:

```
kd -k com:pipe,reconnect,port=\\.\pipe\vmpipe
```

5. Ensure that KD can connect to the kernel. Exit it by pressing Ctrl-B followed by ENTER. **Do not use the 'q' command, as it would resume the kernel.**

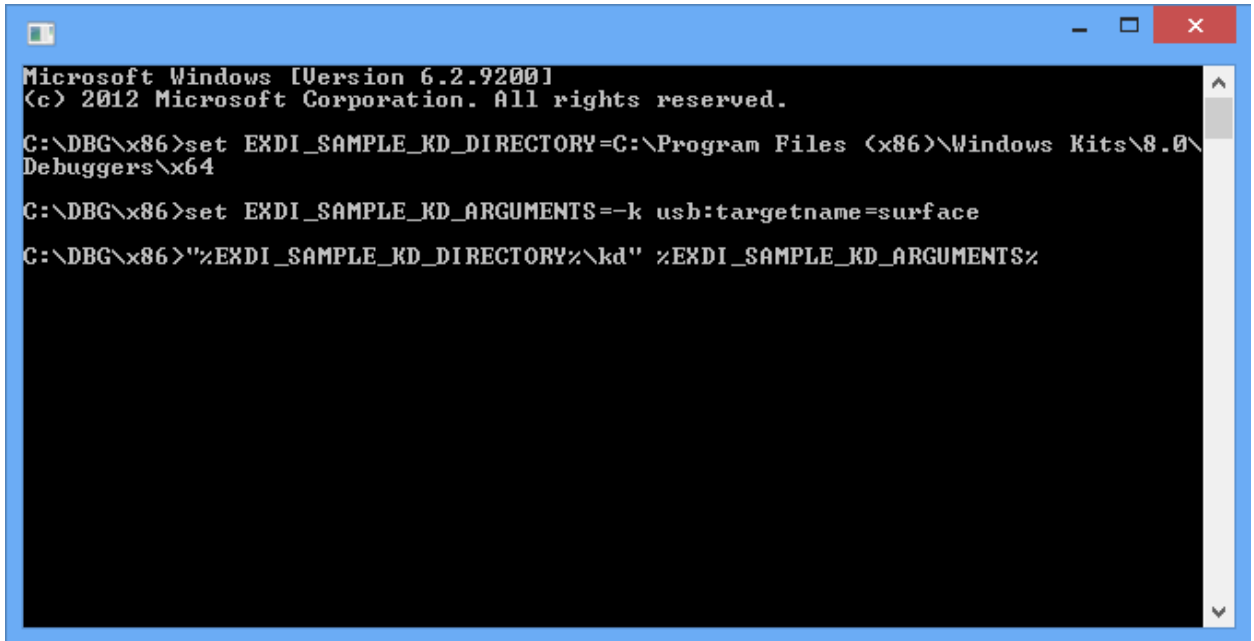


```
Administrator: Command Prompt
*****
Executable search path is:
*****
* Symbols can not be loaded because symbol path is not initialized. *
*
* The Symbol Path can be set by:
*   using the _NT_SYMBOL_PATH environment variable.
*   using the -y <symbol_path> argument when starting the debugger.
*   using .sympath and .sympath+
*****
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntkrnlmp.exe -
Windows 8 Kernel Version 9200 MP (1 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserIS
Built by: 9200.16384.amd64fre.win8_rtm.120725-1247
Machine Name:
Kernel base = 0xfffff801`ce215000 PsLoadedModuleList = 0xfffff801`ce4dfa60
Debug session time: Fri Apr  5 16:25:13.001 2013 (UTC - 7:00)
System Uptime: 0 days 4:12:29.060
Break instruction exception - code 80000003 (first chance)
fffff800`04ac503b cc          int     3
kd> ^B
quit:
C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>
```

6. Set the following environment variables:
 - a. EXDI_SAMPLE_KD_DIRECTORY to the directory containing kd.exe (normally same directory as the current one).
 - b. EXDI_SAMPLE_KD_ARGUMENTS to the arguments used to start kd.exe.

7. Recheck your arguments by running the following command from the command window with the environment variables:

```
"%EXDI_SAMPLE_KD_DIRECTORY%\kd" %EXDI_SAMPLE_KD_ARGUMENTS%
```



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\DBG\x86>set EXDI_SAMPLE_KD_DIRECTORY=C:\Program Files (x86)\Windows Kits\8.0\
Debuggers\x64

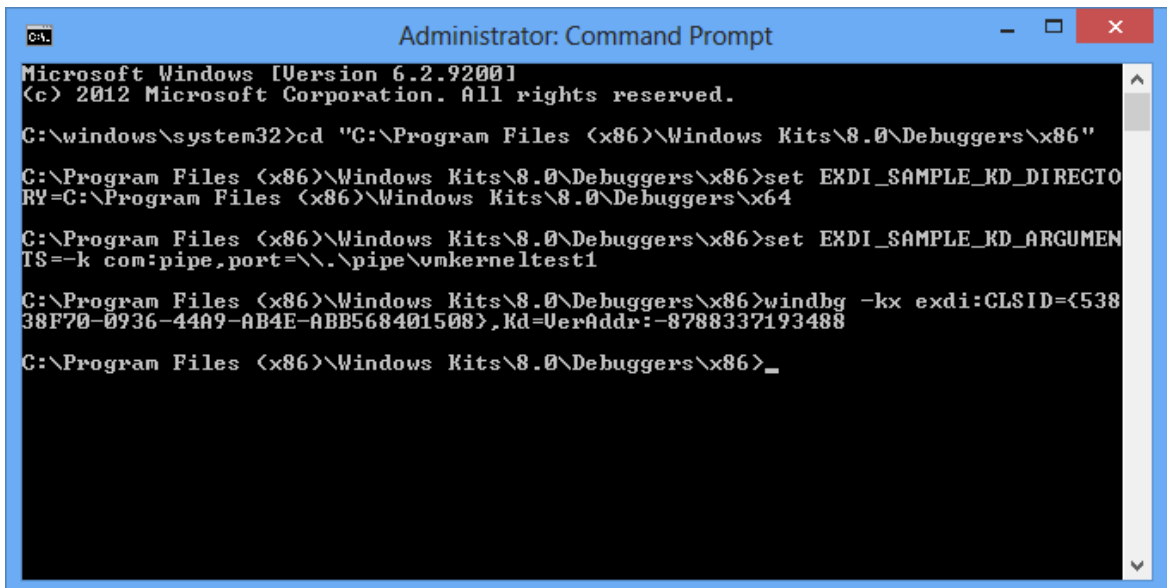
C:\DBG\x86>set EXDI_SAMPLE_KD_ARGUMENTS=-k usb:targetname=surface

C:\DBG\x86>"%EXDI_SAMPLE_KD_DIRECTORY%\kd" %EXDI_SAMPLE_KD_ARGUMENTS%
```

If KD starts successfully, exit it by pressing Ctrl-B followed by Enter.

8. Run WinDbg with the following arguments from the same command prompt:

```
-kx exdi:CLSID={53838F70-0936-44A9-AB4E-ABB568401508},Kd=VerAddr:<Address
of KdVersionBlock in decimal>
```



```
Administrator: Command Prompt

Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\windows\system32>cd "C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86"

C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>set EXDI_SAMPLE_KD_DIRECTO
RY=C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x64

C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>set EXDI_SAMPLE_KD_ARGUMEN
TS=-k com:pipe,port=\\.\pipe\vmkerneltest1

C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>windbg -kx exdi:CLSID={538
38F70-0936-44A9-AB4E-ABB568401508},Kd=VerAddr:-8788337193488

C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>_
```

9. You will see the normal WinDbg window, a kd.exe window and the 'Blind KD Output' window. Go to the normal WinDbg window and ensure that the symbols are loaded:

```
.symfix c:\symbols.net
.reload
```

10. Position the windows so that you can see the 'Blind KD' window while interacting with WinDbg.
11. WinDbg will show that the kernel is stopped at the DbgBreakPoint() call. Run the '?? pTest' command or hover the mouse over pTest to see its value:

```

Command - exDI 'exdi:clsid={53838f70-0936-44a9-ab4e-abb568401508},kd=veradd...
fffff880`04ac4000 fffff880`04acb000 MyDriver1 (private pdb symbols) E:\project\
fffff880`04ae4000 fffff880`04b84000 srv2 (deferred)
fffff960`00058000 fffff960`0044d000 win32k (deferred)
fffff960`00718000 fffff960`00721000 TSDDD (deferred)
fffff960`00873000 fffff960`008a9000 cdd (deferred)

Unloaded modules:
fffff880`04abd000 fffff880`04ac4000 MyDriver1.sys
fffff880`04ab6000 fffff880`04abd000 MyDriver1.sys
fffff880`04aa0000 fffff880`04ab6000 MyDriver1.sys
fffff880`04aa8000 fffff880`04aa0000 MyDriver1.sys
fffff880`04aa1000 fffff880`04aa8000 MyDriver1.sys
fffff880`01ee2000 fffff880`01eef000 dump_ataport.sys
fffff880`01eef000 fffff880`01ef9000 dump_atapi.sys
fffff880`01ef9000 fffff880`01f0d000 dump_dumpfive.sys
fffff880`01e21000 fffff880`01e31000 dam.sys
fffff880`0133f000 fffff880`0134a000 WdBoot.sys
fffff880`01a00000 fffff880`01a0c000 hwpolicy.sys
fffff880`00cf5000 fffff880`00d02000 ApiSetSchema.dll
kd> ?? pTest
char * 0xfffff880`04ac5060
"Hello, World."
kd>

```

12. Note the output in the 'Blind KD' window:

```

Blind KD - please close when done debugging
fffffa80`01f82150 64 00 61 00 6d 00 2e 00-73 00 79 00 73 00 d.a.n...s.y.s.
db fffffa8000cc0880 fffffa8000cc08a7
fffffa80`00cc0880 14 00 16 00 00 00 00 00-f0 5d e6 01 80 fa ff ff .....l.....
fffffa80`00cc0890 00 f0 33 01 80 f8 ff ff-00 a0 34 01 80 f8 ff ff .....3.....4.....
fffffa80`00cc08a0 8d 4d c3 aa 06 2f ce 01 .M.../..
db fffffa8001e65df0 fffffa8001e65e03
fffffa80`01e65df0 57 00 64 00 42 00 6f 00-6f 00 74 00 2e 00 73 00 W.d.B.o.o.t...s.
fffffa80`01e65e00 79 00 73 00 y.s.
db fffffa8000cc0858 fffffa8000cc087f
fffffa80`00cc0858 18 00 1a 00 00 00 00 00-20 db e3 01 80 fa ff ff .....
fffffa80`00cc0868 00 00 a0 01 80 f8 ff ff-00 c0 a0 01 80 f8 ff ff .....
fffffa80`00cc0878 99 b2 87 aa 06 2f ce 01 ...../..
db fffffa8001e3db20 fffffa8001e3db37
fffffa80`01e3db20 68 00 77 00 70 00 6f 00-6c 00 69 00 63 00 79 00 h.w.p.o.l.i.c.y.
fffffa80`01e3db30 2e 00 73 00 79 00 73 00 .s.y.s.
db fffffa8000cc0830 fffffa8000cc0857
fffffa80`00cc0830 20 00 22 00 00 00 00 00-e0 4a cb 00 80 fa ff ff ..".....J.....
fffffa80`00cc0840 00 50 cf 00 80 f8 ff ff-00 20 d0 00 80 f8 ff ff ..P.....
fffffa80`00cc0850 e3 6a c5 a9 06 2f ce 01 ..j.../..
db fffffa8000cb4ae0 fffffa8000cb4aff
fffffa80`00cb4ae0 41 00 70 00 69 00 53 00-65 00 74 00 53 00 63 00 a.p.i.s.e.t.s.c.
fffffa80`00cb4af0 68 00 65 00 6d 00 61 00-2e 00 64 00 6c 00 6c 00 h.e.m.a...d.l.l.
db fffffa8000cc0fd8 fffffa8000cc0fff
fffffa80`00cc0fd8 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffffa80`00cc0fe8 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffffa80`00cc0ff8 00 00 00 00 00 00 00 00 .....
db fffff880047f7850 fffff880047f7857
fffff880`047f7850 60 50 ac 04 80 f8 ff ff 'P.....
db fffff880047f7850 fffff880047f7857
fffff880`047f7850 60 50 ac 04 80 f8 ff ff 'P.....
db fffff88004ac5060 fffff88004ac5fff
fffff880`04ac5060 48 65 6c 6c 6f 2c 20 57-6f 72 6c 64 0a 00 cc cc Hello, World....
fffff880`04ac5070 25 73 00 00 00 00 00 00-00 00 00 00 00 00 00 00 %s.....
fffff880`04ac5080 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50a0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50b0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50c0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50d0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50e0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac50f0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
fffff880`04ac5180 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

```

When you entered the '?? pTest' command, WinDbg used the PDB symbols and the loaded module structures in the Windows kernel to compute the address of the pTest variable (**0xfffff880047f7850** in this example). It then asked our EXDI server to read 8 bytes at that memory location. Then WinDbg interpreted those bytes as a pointer and read the rest of the page containing the string to show it to the user in a more informative way. The EXDI server did not participate in any symbol-related activities – it simply fetched the raw memory contents requested by WinDbg and WinDbg interpreted it.

13. You can run other WinDbg commands and observe how they are translated into memory reading commands handled by the EXDI server. When done, exit WinDbg, and forcibly close the 'Blind KD' and KD.EXE windows.

Live EXDI sample

Once you finished trying out the static EXDI server, replace the CLSID in the WinDbg command line to {67030926-1754-4FDA-9788-7F731CBDAE42}. This will activate a more advanced sample server that supports running the target, setting breakpoints and stepping through the code.

Developing your own EXDI server

In order to add support for your JTAG hardware to WinDbg, all you need to do is create an EXDI server implementing basic operations such as memory reading.

It is recommended to start with modifying the static EXDI sample (`CStaticExdiSampleServer` class). To get minimum functionality you will need to change the following methods:

- `CStaticExdiSampleServer::ReadVirtualMemory()`
- `CStaticExdiSampleServer::GetContext()`
- `CStaticExdiSampleServer::GetRunStatus()`